To Coalesce or Not To Coalesce

Khaled Salah
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
Email: salah@kfupm.edu.sa

Abstract

System performance of Gigabit network hosts can severely be degraded due to interrupt overhead caused by heavy incoming traffic. One of the most popular solutions to mitigate such overhead is interrupt coalescing in which a single interrupt is generated for multiple incoming packets. This is opposed to normal interruption in which an interrupt is generated for every incoming packet. In this paper we investigate the performance of interrupt coalescing analytically and compare it with that of normal interruption. We consider two types of coalescing (viz. count-based and time-based). The system performance is studied in terms of throughput, CPU availability for user applications, latency, and packet loss.

KEYWORDS: High-Speed Networks, Operating Systems, Interrupts, Interrupt Coalescing, Modeling and Analysis, Performance Evaluation.

1 Introduction

Under heavy traffic load, such as that of Gigabit networks, the performance of interrupt-driven systems can be degraded significantly resulting in a poor host performance perceived by the user. Every hardware interrupt for every incoming packet is associated with context switching of saving and restoring processor's state as well as potential cache/TLB pollution. More importantly, interrupt-level handling has absolute priority over all other tasks by definition. If interrupt rate is high enough, the system will spend all of its time responding to interrupts, and hence the system throughput will drop to zero. This situation is called *receive livelock* [1]. In this situation, the system is not deadlocked but causing tasks scheduled at a lower priority to starve.

A number of solutions has been proposed in the literature [1-13] to mitigate interrupt overhead and improve OS performance. Some of these solutions include interrupt coalescing, OS-bypass protocol, zero-copying, jumbo frames, polling, pushing some or all protocol processing to hardware, etc. One of the most popular solutions to mitigate the interrupt overhead for Gigabit network hosts is interrupt coalescing. In recent years, almost all network adapters or network interface cards (NICs) are manufactured to have interrupt coalescing (IC). IC is a feature in which the NIC generates a single interrupt for a group of incoming packets. This is opposed to normal interruption in which the NIC generates an interrupt for every incoming packet.

Although interrupt coalescing is an important feature that is widely available to mitigate interrupt overhead and improve performance, little research has been conducted to study its performance. In [2], a time-based interrupt coalescing was studied using NIC emulation. The OS provided overload conditions to the NIC to adjust the coalescing time. In [5,6], the impact of hosts using interrupt coalescing on the overall bandwidth and latency of IP networks was investigated experimentally. In [7], the performance of interrupt coalescing was analyzed using an experiment that consisted of modifying the Linux kernel of a gateway.

In this paper, we investigate *analytically* the system performance when employing interrupt coalescing. We address the performance of both types of coalescing (viz. count-based IC and time-based IC). We also compare their performance with the performance of normal interruption. The analytical models presented in this paper are based on queueing theory and Markov process. The host performance is studied in terms of system throughput, system latency, host saturation point and system stability condition, CPU utilizations of ISR (Interrupt Service Routine) handling and protocol processing, and CPU availability for user applications.

The rest of the paper is organized as follows. Section 2 presents analytical models that capture the coalescing behavior and study the performance of Gigabit Ethernet hosts. Section 3 shows numerical examples to compare and validate analysis. Finally, Section 5 concludes the study and identifies future work.

2 Analysis

With almost all today's Gigabit NICs and under normal interruption, an incoming packet gets transferred (or DMA'd) through the PCI bus from the NIC to the protocol processing buffer of the kernel. After the packet has been successfully DMA'd, the NIC generates an interrupt to notify the kernel to start protocol processing on the incoming packet. Protocol processing typically involves TCP/IP processing of the incoming packet and delivering it to user applications. During protocol processing other packets may arrive and get queued. Protocol processing time is affected by the interrupts of incoming packets. Interrupt handling has an absolute priority over protocol processing. If an interrupt occurs during protocol processing, the protocol processing will be disrupted or preempted (i.e., protocol processing will stall until the completion of interrupt handling).

2.1 Ideal and Normal Interruption

In previous work [14,15], we presented analytical models to study two types of interrupt handling schemes (viz. ideal scheme and normal interruption). In ideal scheme, the overhead involved in generating interrupts is totally ignored. The ideal scheme gives the best performance that can possibly be obtained when employing interrupts, thus serving as a reference (or a benchmark) to compare with. In normal interruption, every incoming packet causes an interrupt. Closed-form solutions for a number of performance metrics can be found in [14,15].

2.2 Interrupt-Coalescing

There are two types of interrupt coalescing to mitigate the rate of interrupts (viz. count-based and time-based). In this section, we first present analysis for count-based IC and then discuss the analysis for time-based IC. In

count-based IC, the NIC generates an interrupt when a predefined number of packets has been received. In time-based IC, the NIC waits a predefined time period before generating an interrupt. During this time period, multiple packets can be received. It is very important to recognize that the time period gets restarted only when the previous time period has expired and a fresh packet has been received.

Our analytical approach is based on first determining the portion of CPU power (or CPU utilization) consumed by interrupt handling. A Markov process is used to compute such utilization. Knowing the CPU utilization of interrupt handling, one can then find the mean effective protocol processing rate. The mean effective protocol processing rate is actually the protocol processing rate taking into account the disruption factor of interrupt handling. Lastly, a discrete-state continuous-time Markov process is used to model a finite-buffer queueing system with this mean effective processing rate. In addition, the Markov process captures the coalescing behavior.

2.2.1 Count-based IC

For comparison purposes, we will use the same assumptions and notations presented in [14,15]. We assume Poisson incoming traffic, fixed packet sizes, and exponential times for interrupt handling and protocol processing.

Let λ denote the mean incoming packet arrival rate,

μ denote the mean protocol processing rate carried out by the kernel, and thus $1/\mu$ becomes the average time the system takes to process the incoming packet and deliver it to the user application. This time includes primarily the network protocol stack processing carried out by the kernel, excluding any time disruption due to interrupt handling, and

$1/r$ denote the mean ISR or interrupt handling time (i.e., the interrupt service routine time for handling incoming packets). $1/r$ basically includes the interrupt-context switching overhead as well as the ISR handling. The main function of ISR handling is to notify the kernel to start protocol processing of the received packet.

τ denote the coalescing parameter for the predefined number of packets to be coalesced before initiating an interrupt.

Thus, the interrupt frequency or rate is mitigated to $I_{freq} = \frac{\lambda}{\tau}$.

It is important to notice that when $\tau = 1$, an interrupt is generated per packet (i.e., the NIC resorts to normal interrupting with $I_{freq} = \lambda$).

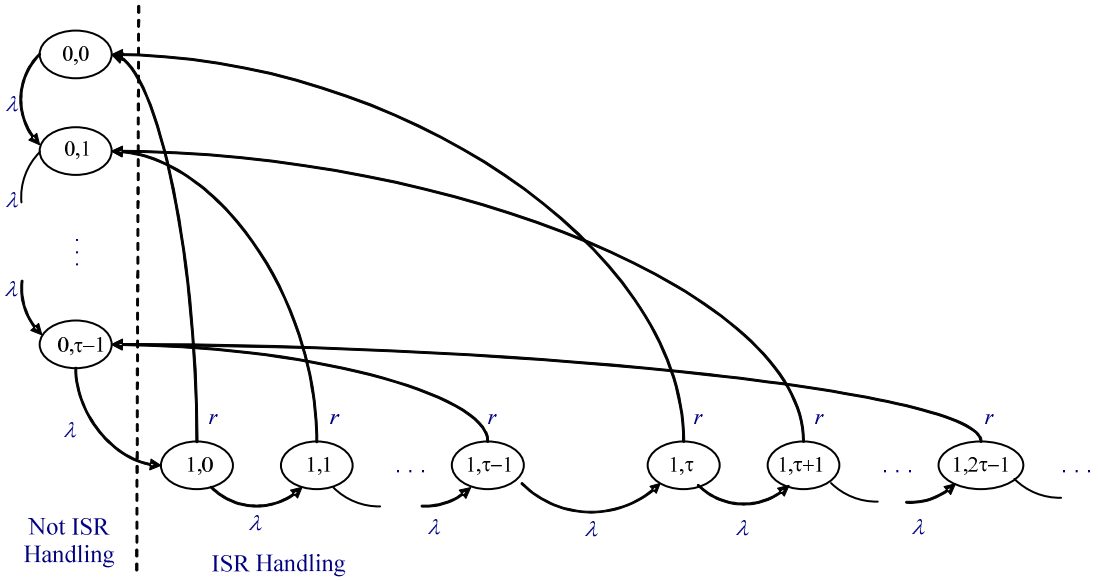


Figure 1. Markov state transition diagram for modeling CPU usage in interrupt coalescing

In order to determine the CPU utilization consumed by interrupt handling when using interrupt coalescing, we use a Markov chain as illustrated in Figure 1.

The state space has states (0,*k*) and states (1,*n*), where

State (0,*k*) represents the state where the CPU is available for protocol processing with $0 \le k < \tau$,

k denotes the number of packet arrivals that are being coalesced before generating an interrupt,

State (1,*n*) represents the state where the CPU is busy handling interrupts with $n \ge 0$, and

n denotes the number of packet arrivals that occurred during ISR handling.

State (1,0) means 0 packet arrived during ISR and the coalescing size is 0. At this point, the ISR may finish with a rate of *r* and thus returns to state (0,0). State (1,1) means 1 packet arrived during ISR and the coalescing size is 1. In this case, the ISR may finish with a rate of *r* and thus returns to state (0,1). State (1,τ-1) means τ-1 packets arrived during ISR and the coalescing size is τ-1. At this point, upon the next packet arrival an interrupt will be generated. If the ISR has finished before this packet arrival, the system will be in state (1,0). However, if the ISR has not finished, this interrupt will be masked off and the system will be in state (1,τ). State (1,τ) indicates that τ packets arrived during ISR and the coalescing size would be (n mod τ) or 0.

Let $p_{n,m}$ be the steady-state probability at state(n,m). A system of difference equations can be derived for the stationary probabilities as follows:

For states (0,*k*), where $0 \le k < \tau - 1$, we have

$$
\left.
\begin{aligned}
&-\lambda p_{0,0} + r p_{1,0} + r p_{1,\tau} + r p_{1,2\tau} + r p_{1,3\tau} + \cdots = 0, \\
&-\lambda p_{0,1} + \lambda p_{0,0} + r p_{1,1} + r p_{1,\tau+1} + r p_{1,2\tau+1} + r p_{1,3\tau+1} + \cdots = 0, \\
&-\lambda p_{0,2} + \lambda p_{0,1} + r p_{1,2} + r p_{1,\tau+2} + r p_{1,2\tau+2} + r p_{1,3\tau+2} + \cdots = 0, \\
&\qquad\qquad \vdots \\
&-\lambda p_{0,k} + \lambda p_{0,k-1} + r p_{1,k} + r p_{1,\tau+k} + r p_{1,2\tau+k} + r p_{1,3\tau+k} + \cdots = 0
\end{aligned}
\right\} \qquad (1)
$$

For states (1,n), where $n \geq 0$, we have

$$
\left.
\begin{aligned}
&-(\lambda + r) p_{1,0} + \lambda p_{0,\tau-1} = 0, \\
&-(\lambda + r) p_{1,1} + \lambda p_{1,0} = 0, \\
&-(\lambda + r) p_{1,2} + \lambda p_{1,1} = 0, \\
&\qquad \vdots \\
&-(\lambda + r) p_{1,n} + \lambda p_{1,n-1} = 0.
\end{aligned}
\right\} \qquad (2)
$$

Let $\beta = \lambda/(\lambda + r)$, then Equations (2) can be simplified to

$$
p_{1,n} = \left(\frac{\lambda}{\lambda + r}\right) p_{1,n-1} = \beta^{n+1} p_{0,\tau-1} \qquad n \geq 0. \qquad (3)
$$

In order to solve Equations (1), we need to express each equation in (1) in terms of $p_{0,\tau-1}$. Then $p_{0,0}$ can be expressed as

$$
\begin{aligned}
\lambda p_{0,0} &= r\left(p_{1,0} + p_{1,\tau} + p_{1,2\tau} + p_{1,3\tau} + \cdots\right) \\
&= r\left(\sum_{n=0}^{\infty} p_{1,n\tau}\right) = r\left(\sum_{n=0}^{\infty} \beta^{n\tau+1} p_{0,\tau-1}\right) = r\left(\frac{\beta}{1-\beta^{\tau}}\right) p_{0,\tau-1}, \\
p_{0,0} &= \left(\frac{1-\beta}{1-\beta^{\tau}}\right) p_{0,\tau-1}.
\end{aligned}
$$

$p_{0,1}$ can be expressed as

$$
\begin{aligned}
\lambda p_{0,1} &= \lambda p_{0,0} + r\left(p_{1,1} + p_{1,\tau+1} + p_{1,2\tau+1} + p_{1,3\tau+1} + \cdots\right) \\
&= \lambda p_{0,0} + r\left(\sum_{n=0}^{\infty} p_{1,n\tau+1}\right) = \lambda p_{0,0} + r\left(\sum_{n=0}^{\infty} \beta^{n\tau+2} p_{0,\tau-1}\right) \\
&= \lambda p_{0,0} + r\left(\frac{\beta^2}{1-\beta^{\tau}}\right) p_{0,\tau-1}, \\
p_{0,1} &= \frac{(1-\beta)(1+\beta)}{1-\beta^{\tau}} p_{0,\tau-1}.
\end{aligned}
$$

In general, $p_{0,k}$ can be expressed as

$$
p_{0,k} = \frac{(1-\beta)(1+\beta^2+\beta^3+\cdots+\beta^k)}{1-\beta^{\tau}} p_{0,\tau-1} \qquad 0 \leq k \leq \tau - 1.
$$

or

$$p_{0,k} = \frac{1-\beta^{k+1}}{1-\beta^{\tau}} p_{0,\tau-1} . \tag{4}$$

To find the value of $p_{0,\tau-1}$, we utilize the law of total probability in which

$$\sum_{k=0}^{\tau-1} p_{0,k} + \sum_{n=0}^{\infty} p_{1,n} = 1 .$$

This can be rewritten as

$$\sum_{k=0}^{\tau-2} p_{0,k} + p_{0,\tau-1} + \sum_{n=0}^{\infty} p_{1,n} = 1 .$$

Since

$$\sum_{n=0}^{\infty} p_{1,n} = \sum_{n=0}^{\infty} \beta^{n+1} p_{0,\tau-1} = \left(\frac{\beta}{1-\beta} \right) p_{0,\tau-1},$$

then, we have

$$p_{0,\tau-1} \left[\sum_{k=0}^{\tau-2} \left(\frac{1-\beta^{k+1}}{1-\beta^{\tau}} \right) + 1 + \frac{\beta}{1-\beta} \right] = 1,$$

or

$$p_{0,\tau-1} = \left[\sum_{k=0}^{\tau-2} \left(\frac{1-\beta^{k+1}}{1-\beta^{\tau}} \right) + 1 + \frac{\beta}{1-\beta} \right]^{-1} = \frac{1-\beta^{\tau}}{\tau} . \tag{5}$$

Substituting Equation (5) into Equation (3) and Equation (4), we get

$$p_{0,n} = \left(\frac{1-\beta^{n+1}}{1-\beta^{\tau}} \right) \cdot \left(\frac{1-\beta^{\tau}}{\tau} \right) = \frac{1-\beta^{n+1}}{\tau} \qquad 0 \le n \le \tau - 1 . \tag{6}$$

$$p_{1,n} = \beta^{n+1} \cdot \left(\frac{1-\beta^{\tau}}{\tau} \right) \qquad n \ge 0 . \tag{7}$$

Since $\sum_{k=0}^{\tau-1} p_{0,k}$ represents the CPU availability for protocol processing, then this term can be expressed as

$$\sum_{k=0}^{\tau-1} p_{0,k} = \frac{\tau - \beta(1-\beta^{\tau})/(1-\beta)}{\tau} . \tag{8}$$

Similarly, $\sum_{n=0}^{\infty} p_{1,n}$ represents the CPU utilization due to ISR handling, U_{ISR}, and can be expressed as

$$\begin{aligned} U_{ISR} &= \sum_{n=0}^{\infty} p_{1,n} = \sum_{n=0}^{\infty} \beta^{n+1} \left(\frac{1-\beta^{\tau}}{\tau} \right), \\ U_{ISR} &= \frac{\beta}{\tau} \left(\frac{1-\beta^{\tau}}{1-\beta} \right). \end{aligned} \tag{9}$$

Effective Protocol Processing Rate. As discussed earlier, interrupt handling has an absolute priority over protocol processing. If an interrupt occurs during protocol processing, the protocol processing will be disrupted

or preempted. Thus, the mean effective protocol processing rate μ' is actually the protocol processing rate μ taking into account the disruption factor of interrupt handling. This disruption factor is essentially U_{ISR}. The mean effective rate μ' can then be expressed in terms of the CPU power percentage available for protocol processing (i.e., $(1-U_{ISR})$). Therefore the effective protocol processing rate for interrupt coalescing can be expressed as

$$\mu' = \mu \cdot (1 - U_{ISR}),$$
$$\mu' = \mu \cdot \left(\frac{\lambda(\beta^{\tau} - 1) + \tau r}{\tau r} \right), \tag{10}$$

where $\beta = \lambda/(\lambda + r)$.

Note that this effective service time (with a mean $1/\mu'$) is exponentially distributed as the service time (with a mean of $1/\mu$) is assumed to be exponential.

Figure 2 shows a discrete-state continuous-time Markov process that models a finite-buffer queueing system with service rate of μ'. The model captures the coalescing behavior. We use a Markov chain of state space $S = \{(n,m),\ n \in \{0,1\},\ 0 \le m < \infty\}$, where

m denotes the number of packets in the protocol processing buffer.

n denotes the protocol processing status with either 0 or 1. 0 means that protocol processing is idle, and 1 means protocol processing is active. When *n*=0, the protocol processing is idle waiting for more packets to be coalesced before the processing of packets starts. Protocol processing starts when the NIC generates an interrupt after τ packets have been received. When *n*=1, the protocol processing is active serving packets.

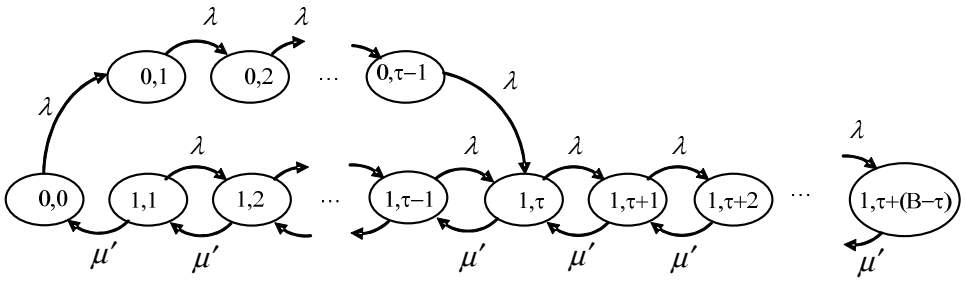


Figure 2. Markov state transition diagram for modeling a finite-buffer queueing system with coalescing

We derive the stationary probabilities for this model by finding the balance equation for each state at a time. At state (0,0), we have

$$-\lambda p_{0,0} + \mu' p_{1,1} = 0 \qquad \Rightarrow \qquad p_{1,1} = \frac{\lambda}{\mu'} p_{0,0}.$$

At state (0,1), we have

$$-\lambda p_{0,1} + \lambda p_{0,0} = 0 \qquad \Rightarrow \quad p_{0,1} = p_{0,0}.$$

Similarly for states (0,2), (0,3), …, and (0, τ-1), we get

$$p_{0,k} = p_{0,0} \qquad 1 \le k \le \tau - 1. \tag{11}$$

At state (1,1), we have

$$-(\lambda+\mu')\,p_{1,1}+\mu' p_{1,2}=0,$$

$$p_{1,2}=\left(\frac{\lambda}{\mu'}\right)^2 p_{0,0}+\frac{\lambda}{\mu'}p_{0,0}.$$

At state (1,2), we have

$$-(\lambda+\mu')\,p_{1,2}+\lambda p_{1,1}+\mu' p_{1,3}=0,$$

$$p_{1,3}=\left(\frac{\lambda}{\mu'}\right)^3 p_{0,0}+\left(\frac{\lambda}{\mu'}\right)^2 p_{0,0}+\frac{\lambda}{\mu'}p_{0,0}.$$

At state (1,3), we have

$$-(\lambda+\mu')\,p_{1,3}+\lambda p_{1,2}+\mu' p_{1,4}=0,$$

$$p_{1,4}=\left(\frac{\lambda}{\mu'}\right)^4 p_{0,0}+\left(\frac{\lambda}{\mu'}\right)^3 p_{0,0}+\left(\frac{\lambda}{\mu'}\right)^2 p_{0,0}+\frac{\lambda}{\mu'}p_{0,0}.$$

Thus, at state (1,n) where $1\le n\le \tau-1$, we have

$$-(\lambda+\mu')\,p_{1,n}+\lambda p_{1,n-1}+\mu' p_{1,n+1}=0,$$

$$\mu' p_{1,n+1}=(\lambda+\mu')\left[\left(\frac{\lambda}{\mu'}\right)^n p_{0,0}+\left(\frac{\lambda}{\mu'}\right)^{n-1} p_{0,0}+\cdots+\frac{\lambda}{\mu'}p_{0,0}\right]$$

$$-\lambda\left[\left(\frac{\lambda}{\mu'}\right)^{n-1} p_{0,0}+\left(\frac{\lambda}{\mu'}\right)^{n-2} p_{0,0}+\cdots+\frac{\lambda}{\mu'}p_{0,0}\right],$$

$$p_{1,n+1}=\left(\frac{\lambda}{\mu'}\right)^{n+1} p_{0,0}+\left(\frac{\lambda}{\mu'}\right)^{n} p_{0,0}+\cdots+\frac{\lambda}{\mu'}p_{0,0}. \qquad (12)$$

At state$(1,\tau)$, we have

$$-(\lambda+\mu')\,p_{1,\tau}+\lambda p_{0,\tau-1}+\lambda p_{1,\tau-1}+\mu' p_{1,\tau+1}=0,$$

$$\mu' p_{1,\tau+1}=(\lambda+\mu')\left[\left(\frac{\lambda}{\mu'}\right)^{\tau} p_{0,0}+\left(\frac{\lambda}{\mu'}\right)^{\tau-1} p_{0,0}+\cdots+\frac{\lambda}{\mu'}p_{0,0}\right]$$

$$-\lambda\left[\left(\frac{\lambda}{\mu'}\right)^{\tau-1} p_{0,0}+\left(\frac{\lambda}{\mu'}\right)^{\tau-2} p_{0,0}+\cdots+\frac{\lambda}{\mu'}p_{0,0}\right]-\lambda p_{0,0},$$

$$p_{1,\tau+1}=\left(\frac{\lambda}{\mu'}\right)^{\tau+1} p_{0,0}+\left(\frac{\lambda}{\mu'}\right)^{\tau} p_{0,0}+\cdots+\left(\frac{\lambda}{\mu'}\right)^{2} p_{0,0}.$$

Thus

$$p_{1,\tau+n} = \left(\frac{\lambda}{\mu'}\right)^{\tau+n} p_{0,0} + \left(\frac{\lambda}{\mu'}\right)^{\tau+n-1} p_{0,0} + \cdots + \left(\frac{\lambda}{\mu'}\right)^{n+1} p_{0,0} \text{ where } n \geq 0 . \quad (13)$$

Now if we let $\rho_{IP} = \lambda / \mu'$, Equation (12) and Equation (13) can be simplified to

$$\begin{aligned} p_{1,n} &= p_{0,0} \sum_{i=1}^{n} \rho_{IP}^{i} \text{ where } 1 \leq n \leq \tau - 1. \\ p_{1,\tau+n} &= \sum_{i=1}^{\tau} \rho_{IP}^{n+i} p_{0,0} = \rho_{IP}^{n} p_{0,0} \sum_{i=1}^{\tau} \rho_{IP}^{i} \text{ , where } n \geq 0 . \end{aligned} \quad (14)$$

To find $p_{0,0}$, we utilize the law of total probability in which

$$p_{0,0} + p_{0,1} + \cdots + p_{0,\tau-1} + p_{1,\tau} + p_{1,\tau+1} + p_{1,\tau+2} + \cdots + p_{1,\tau+(B-\tau)} = 1,$$

or

$$p_{0,0} \sum_{n=0}^{\tau-1} 1 + p_{0.0} \sum_{n=1}^{\tau-1} \sum_{i=1}^{n} \rho_{IP}^{i} + p_{0,0} \sum_{k=1}^{\tau} \rho_{IP}^{k} \sum_{n=0}^{B-\tau} \rho_{IP}^{n} = 1 .$$

Solving for $p_{0,0}$, we get

$$p_{0,0} = \left[\tau + \sum_{n=1}^{\tau-1} \sum_{i=1}^{n} \rho_{IP}^{i} + \sum_{k=1}^{\tau} \rho_{IP}^{k} \sum_{n=0}^{B-\tau} \rho_{IP}^{n} \right]^{-1} . \quad (15)$$

This can be further simplified as

$$p_{0,0} = \frac{\rho_{IP}^{\tau} (\rho_{IP} - 1)^2}{\rho_{IP}^{\tau} (\tau - \tau\rho_{IP} + \rho_{IP}^{B+2}) - \rho_{IP}^{B+2}} . \quad (16)$$

CPU Utilization and Availability. The percentage of CPU power *V* (which is available for user applications) is basically the probability when there is no ISR handling and kernel protocol processing is idle. Hence, the CPU availability for user applications can be expressed as

$$V = (1 - U_{ISR}) \cdot p_0 , \quad (17)$$

where p_0 is the idleness of protocol processing (or probability of no queueing). Please note that protocol processing is idle whenever it is at state (0,0), (0,1), …, or (0,τ-1). Thus the idleness of protocol processing can be expressed as

$$p_0 = p_{0,0} \sum_{n=0}^{\tau-1} 1 = p_{0,0} \, \tau . \quad (18)$$

The CPU utilization for ISR handling U_{ISR} is given by Equation (9). The CPU utilization for protocol processing U_{IP} can be expressed in three forms which are mathematically equivalent. The first form is $U_{IP} = U_{ISR+IP} - U_{ISR}$. This can be simplified using substitution of Equations (9), (17), and (18) to

$U_{IP} = (1 - U_{ISR})(1 - p_{0,0}\ \tau)$. The second form is to express U_{IP} as the probability of no ISR handling and protocol processing idleness. That is,

$$U_{IP} = (1 - U_{ISR})(1 - p_{0,0}\ \tau). \tag{19}$$

And the third form is to observe that U_{IP} can also be derived by summing up all the state probabilities of (1,*n*) where $1 \le n \le B$, as

$$U_{IP} = p_{0.0} \sum_{n=1}^{\tau-1} \sum_{i=1}^{n} \rho_{IP}^{i} + p_{0,0} \sum_{k=1}^{\tau} \rho_{IP}^{k} \sum_{n=0}^{B-\tau} \rho_{IP}^{n}\ . \tag{20}$$

With simplification, it can be proven that Equation (20) is equivalent to Equation (19).

Mean System Throughput and Loss Probability. The mean system throughput γ is basically the departure rate due to protocol processing. γ can be expressed in three forms, which are all mathematically equivalent. Three forms are given for verification purposes.

(i) γ can be expressed as

$$\gamma = \mu'(1 - p_0)\,, \tag{21}$$

where p_0 is expressed by Equation (18), or

(ii) γ can be expressed as

$$\gamma = \sum_{n=1}^{B} \mu'\ p_{1,n} = \mu' \left(1 - \sum_{n=0}^{\tau-1} p_{0,n}\right) = \mu'(1 - \tau\ p_{0,0}),$$

where $p_{0,0}$ is given by Equation (16), or

(iii) γ can be expressed as the effective arrival rate λ' which is $\lambda(1 - P_{loss})$. Therefore,

$$\gamma = \lambda(1 - P_{loss})\,,$$

where P_{loss} is the loss probability for a protocol buffer of size *B*. P_{loss} is basically $p_{1,\tau+(B-\tau)}$ or $p_{1,B}$ and can be expressed using Equation (14) as

$$p_{1,\tau+(B-\tau)}\ = \rho_{IP}^{B-\tau}\, p_{0,0} \sum_{i=1}^{\tau} \rho_{IP}^{i}\,,$$

And thus,

$$p_{loss}\ = p_{0,0} \left(\frac{\rho_{IP}^{B-\tau+1} - \rho_{IP}^{B+1}}{1 - \rho_{IP}} \right).$$

Saturation Point. The saturation or the cliff point in interrupt coalescing occurs when

$$V = 0 \quad or \quad \rho_{IP} = 1 \quad or \quad \lambda = \mu'. \tag{22}$$

This relation can be derived by simply setting V in Equation (17) to zero resulting in $p_0 = 0$. Substituting for p_0 in Equation (18) and then Equation (16), we get $\rho_{IP} = 1$. Solving for λ by itself in $\lambda = \mu'$ is somewhat complicated due to the power term in Equation (10). However, the saturation condition can be solved numerically and can be easily identified graphically, as will be demonstrated in the numerical examples given in Section 3.

Mean System Latency. The mean system latency per packet in interrupt coalescing is affected by both ISR handling and protocol processing. An incoming packet experiences a delay due to interrupt handling and due to the delay of protocol processing. The mean system delay is therefore decomposed into the sum of the mean delay of interrupt handling and the mean delay of protocol processing. Hence the total mean system delay, $E(r)$, can be expressed as

$$E(r) = E_{ISR}(r) + E_{IP}(r),$$

where $E_{ISR}(r)$ is the mean delay due to ISR and $E_{IP}(r)$ is mean delay due to protocol processing. $E_{ISR}(r)$ is basically $1/r$. This is so due to the nature of servicing packets during ISR handling. The mean ISR handling time for one or multiple packets is practically the same (i.e., $1/r$). This delay can also be computed using the Markov chain model depicted in Figure 1 by composing all the states of "Not ISR Handling" into one single state.

Therefore, the mean system delay can be expressed as

$$E(r) = \frac{1}{r} + \frac{E_{IP}(n)}{\lambda'}. \tag{23}$$

The mean delay caused by protocol processing, $E_{IP}(r)$, can be expressed as

$$E_{IP}(r) = \frac{E_{IP}(n)}{\lambda'}.$$

where γ is the mean effective arrival rate is expressed in Equation (21). $E_{IP}(n)$ is the average number of packets encountered due to protocol processing and can be expressed as

$$\begin{aligned} E_{IP}(n) &= \sum_n n\, p_{i,n} = \sum_{n=1}^{\tau-1} n \times (p_{0,n} + p_{1,n}) + \sum_{n=0}^{B-\tau} (n+\tau) \times p_{1,\tau+n} \\ &= \sum_{n=1}^{\tau-1} n \times \left(p_{0,0} + p_{0,0} \sum_{i=1}^{n} \rho_{IP}^i \right) + \sum_{n=0}^{B-\tau} (n+\tau) \times \left(p_{0,0}\, \rho_{IP}^i \sum_{i=1}^{\tau} \rho_{IP}^i \right) \end{aligned}$$

This can be derived and simplified further to

$$E_{IP}(n) = \frac{2\rho_{IP}^{B+2}\left(1 - B(\rho_{IP} - 1)\right) \; - \; \tau\rho_{IP}^{\tau}\left(\rho_{IP} - 1\right)\left(1 + \tau(\rho_{IP} - 1) - 3\rho_{IP}\right) \; - \; 2\rho_{IP}^{B+\tau+2}\left(1 - B(\rho_{IP} - 1)\right)}{2(\rho_{IP} - 1)\left(\rho_{IP}^{\tau}\left(\tau - \tau\rho_{IP} + \rho_{IP}^{B+2}\right) - \rho_{IP}^{B+2}\right)}.$$

Special Case. There is special case of interest that can be used to verify our analysis and mathematical derivation. The special case is when $\tau = 1$. With mathematical substitution, manipulation and simplification, it can be proven that all above derived equations resort exactly to the corresponding ones of normal interruption presented in [15]. In addition, all the numerical examples given in Section 3 show an exact matching is achieved between the curves for normal interruption and those of coalescing with $\tau = 1$.

2.2.2 Time-based IC

In time-based IC, the NIC waits a predefined time period before it generates an interrupt. During this time period multiple packets can be received. The time period gets restarted only when the previous time period has expired and a fresh packet is received. Modeling such a time-based IC is complex. However, one can model it approximately by transforming it into count-based IC model.

Let *T* denote the fixed predefined time period that the NIC has to wait for before generating an interrupt upon the reception of a packet. The average number of packets arriving during *T* can be given by $\tau = \lambda T$ using Little's law[1]. Since τ has to be an integer, $\tau = \lceil \lambda T \rceil$. Note that $\tau = 1$ when $1/\lambda > T$ (i.e., the case of normal interruption). This is intuitively sensible because when $1/\lambda > T$, an interrupt will be generated for every packet. It is worth noting here that the coalescing parameter τ for this approximated time-based IC model is changing and is dependent on the arrival rate λ. For count-based IC, τ is fixed. As will be demonstrated in Section 3, this approximated analytical model give adequate matching numeric results to those of a discrete-even simulation.

Using this approximated analytical model for time-based IC, all performance equations of count-based IC can be applied with the exception of mean system latency when $1/\lambda > T$. When $1/\lambda \le T$, the mean latency can be properly expressed by Equation (23). However for the case when $1/\lambda > T$, Equation (23) has to take into account adding an extra delay. The protocol processing of an incoming packet arriving after the expiration of the polling period *T* will be delayed by an entire period *T* if protocol processing is idle. The average of this extra delay can be expressed simply as *T* given that the protocol processing is idle (i.e., $p_0 \cdot T$). p_0 is the probability of no queueing and is expressed in Equation (18). Therefore, for $1/\lambda > T$ the mean system latency $E(r)$ can be expressed as

---

[1] Another direct derivation of τ is using the general equation $\sum_n n \times p_n$, where *n* is the number of packets arrived during *T* and $p_n = \frac{(\lambda T)^n}{n!} e^{-(\lambda T)}$. Thus, $\tau = e^{-(\lambda T)} \sum_{n=1}^{\infty} n \times \frac{(\lambda T)^n}{n!} = \lambda T$.

$$E(r) = p_0 \cdot T + \frac{1}{r} + \frac{E_{IP}(n)}{\lambda'} . \quad (24)$$

It is important to note that for the count-based IC model with $\tau = 1$, there is no extra delay incurred as the interrupt is generated immediately upon packet arrival.

2.3 Simulation

We developed a discrete-event simulation (DES) model in order to validate our analytical models. The implementation of the DES was written in C language. The assumptions of analysis in Section 2 were used. Guidelines given in [16] were followed carefully. The simulation was automated to produce independent replications with different initial seeds that were one million apart. The initial seeds for the simulation model random variables, within each replication, were chosen to be five million apart. During the simulation run, we checked for overlapping streams and ascertain that such a condition did not exist. The simulation was terminated when achieving a precision of no more than 10% of the mean with a confidence of 90%. We implemented dynamically the *replication/deletion* approach for means [16]. The length of the initial transient period using the MCR (Marginal Confidence Rule) heuristic developed in [17] was computed. Each replication run lasts for five times the length of the initial transient period. Analytical and simulation results, as will be demonstrated in Section 3, were very much in line.

3 Numerical Examples

In this section, the results of analysis and simulation are reported. Numerical results are given for key performance indicators. These indicators include mean system throughput, CPU availability, latency, and packet loss. We compare the performance for all interrupt schemes under study which include the ideal system, normal interruption, count-based IC, and time-based IC.

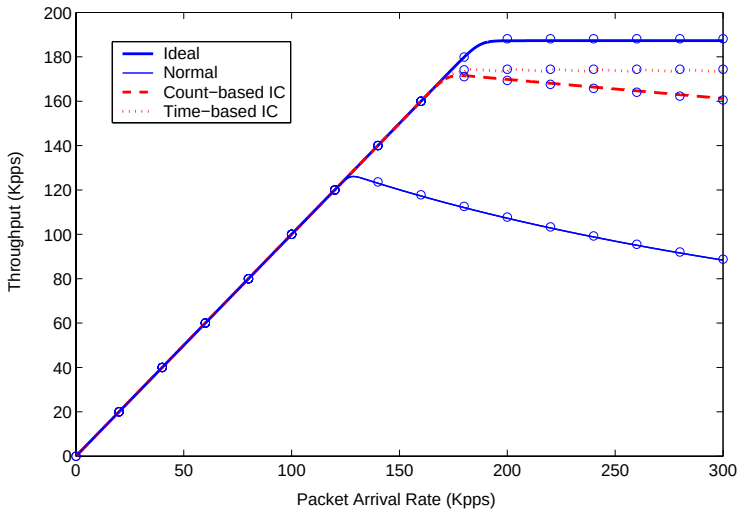
A mean interrupt handling time ($1/r$) of 3.73 μs and a TCP processing ($1/\mu$) of 5.34 μs were used. In all of our examples, a kernel's protocol processing buffer B of a size of 1000 packets is used. These values are realistic and selected based on experimental results involving a modern 2.53GHz Pentium-IV machine [18]. Coalescing parameters of $\tau = 1$ and $\tau = 8$ are used for count-based IC and $T = 0$ and $T = 50$ μs are used for time-based IC.

Figures 3 plots the mean system throughput, CPU availability, mean system latency at low load, mean system latency at high load, and packet loss probability as a function of system load. DES simulation results are plotted with empty circles. The figure exhibits a very close agreement between simulation and analysis results, and thus exhibiting the soundness of our analysis.

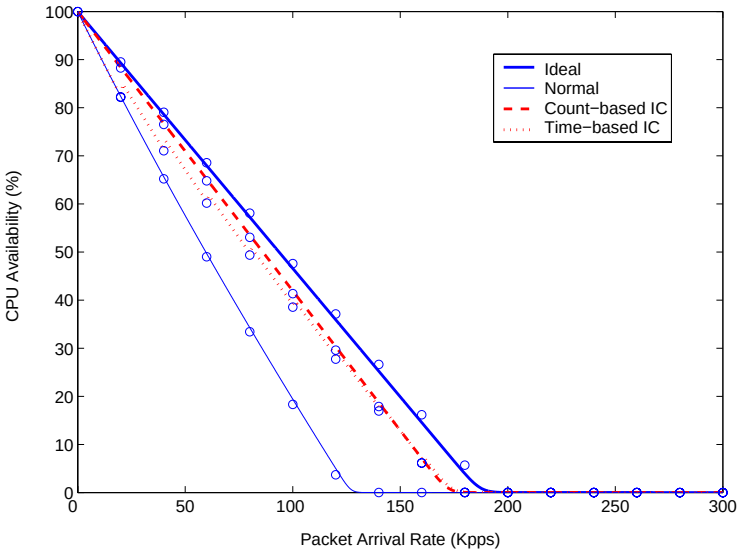
From the figures, it is observed that the maximum throughput occurs at 187 Kpps. For normal interruption, it can be noted that the saturation or cliff point for the system occurs at 127 Kpps. At this point, the corresponding CPU utilization (for ISR handling plus protocol processing) is at 100% resulting in a CPU availability of zero. Figure 3(a) shows that as the arrival rate increases after the cliff point the system throughput starts to decline. Figure 3(d) shows that the mean system delay continues to increase after reaching the saturation point. The decline in the throughput and the increase in latency are due to the fact that the mean effective service rate μ' decreases as the arrival rate increases right after the saturation point. This is very much in line with analysis (as given by Equation 1).

It is depicted from the figures that the throughput improves noticeably in the heavy-load region with coalescing parameters of $\tau > 1$ *or* $T > 0$. The heavy-load region is that region beyond the saturation point of normal interruption. In addition the corresponding CPU availability and packet loss improve. On the other hand, the corresponding system latency is degraded at low load and improved at high load. At low load, the latency is degraded (becomes higher) as packets have to wait longer time to be coalesced.

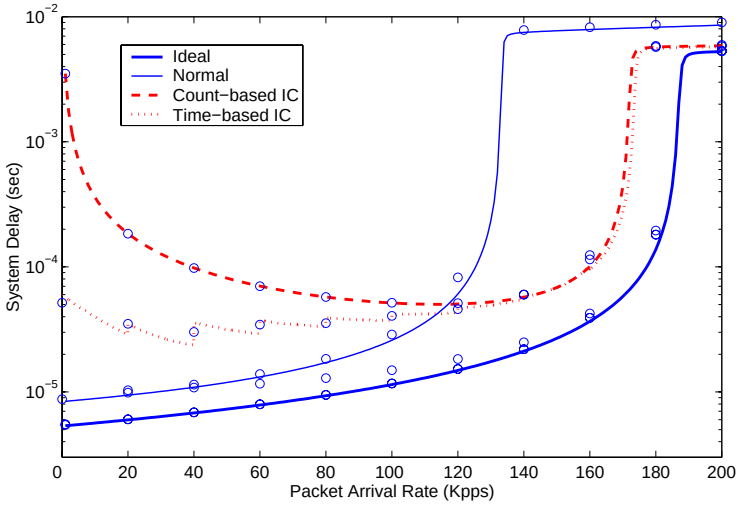
One observation can be made about the coalescing parameters of $\tau = 1$ in the case of count-based IC and $T = 0$ in the case of time-based IC. It was observed that in such cases, both coalescing scheme resort exactly to normal interruption (as expected). Also from the figures, it is depicted that the analysis curves for time-based coalescing are not smooth (more noticeable in Figure 3(b) and 3(c) at very low rate). This is due to the fact that the analysis for time-based IC is performed based on the analysis of count-based IC with the coalescing parameter τ being an integer and approximated by $\lceil \lambda T \rceil$. Thus, τ takes on discrete values and remains unchanged until a different value is produced as λ changes in $\lceil \lambda T \rceil$.



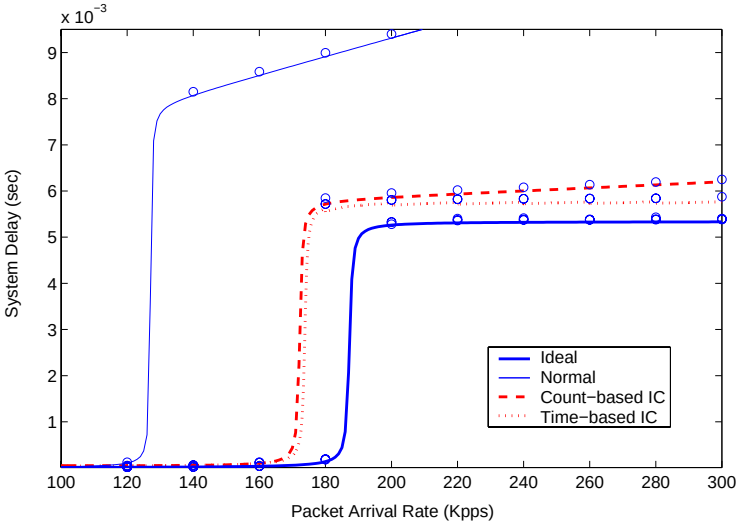
(a)


CPU Availability (%)
Packet Arrival Rate (Kpps)
Ideal
Normal
Count−based IC
Time−based IC

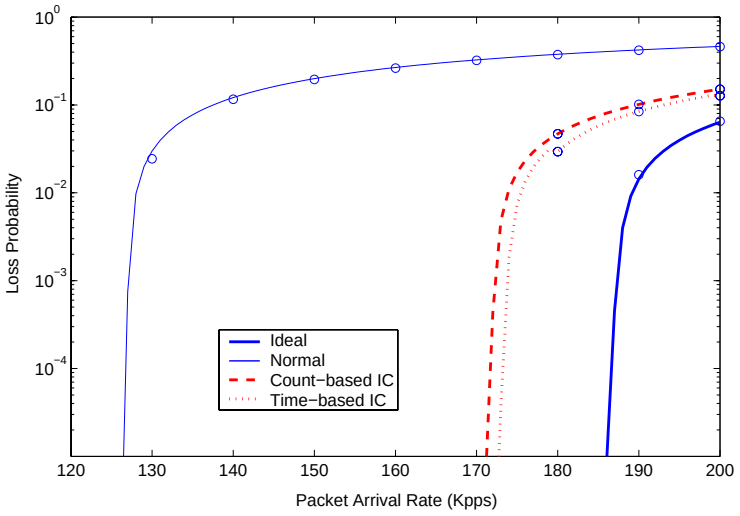
(b)


System Delay (sec)
Packet Arrival Rate (Kpps)
Ideal
Normal
Count−based IC
Time−based IC

(c)


x 10^-3
System Delay (sec)
Packet Arrival Rate (Kpps)
Ideal
Normal
Count−based IC
Time−based IC

(d)



(e)

Figure 3. Key performance indicators in relation to traffic load

Figure 4 shows the impact of selecting different values for those parameters. In general, Figures 4((a)-(c)) show that selecting large values for the count-based coalescing parameter τ can be very detrimental to performance in terms of latency at low load. At very low arrival rate, packets have to wait longer time to be coalesced. The larger the value of τ, the larger the coalescing delay. In time-based coalescing, there is a bound on this coalescing delay, which is the value of *T*. Therefore, if IC to be used (e.g., for hosts that provide non-real time services that can tolerate delay as that of FTP traffic), it is recommended to use time-based IC over count-based IC when hosts are subjected to low or moderate load.

An important observation one can make about the gain in performance in relation to selecting an appropriate value for the coalescing parameters τ and *T*. From the figures, one can observe and conclude that the performance gain (obtained for throughput, CPU availability, or system latency at high load) becomes marginal as τ and *T* take larger values. For example the performance gain when $\tau = 1$ and $\tau = 2$ is more noticeable than that of the performance gain when $\tau = 2$ and $\tau = 3$.

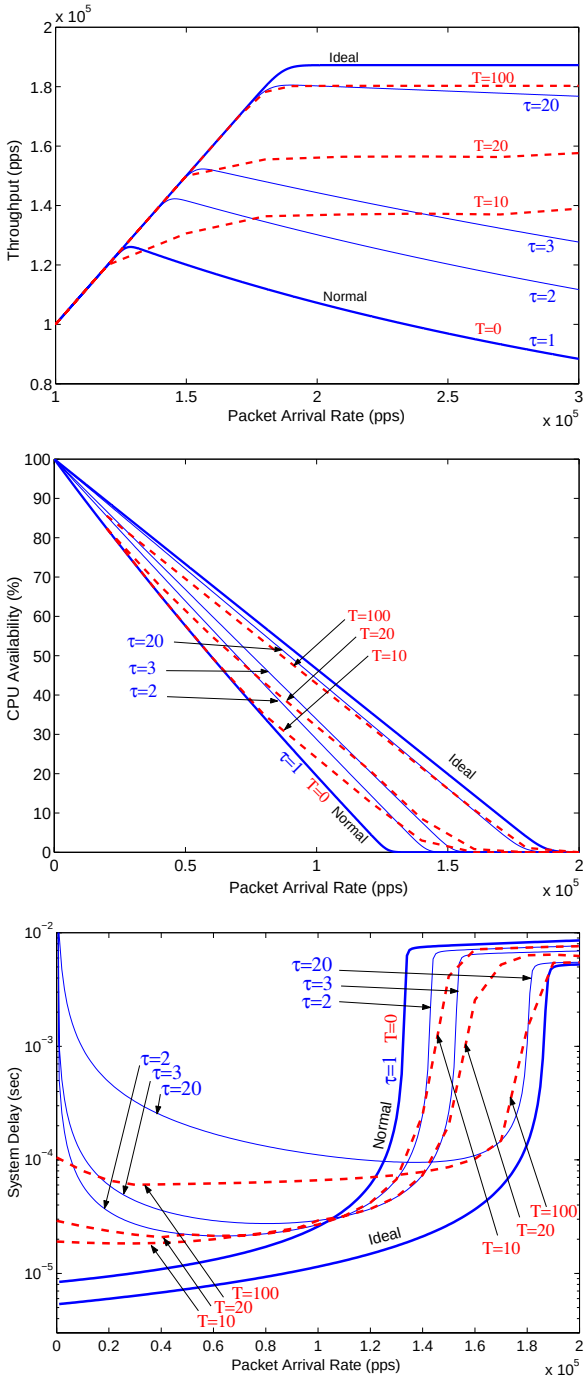


Figure 4. Impact of coalescing parameters on performance

4 Concluding Remarks

We developed analytical models for evaluating the performance of interrupt handling schemes of normal interruption and interrupt coalescing. The analytical models were validated by considering special cases and by simulation. The performance was studied in terms of system throughput, CPU availability, latency, and packet

loss. It was demonstrated that at high traffic load, interrupt coalescing outperforms (in terms of *all* performance indicators) normal interruption. However, at low traffic load, normal interruption outperforms (*only* in terms of latency) interrupt-coalescing mode. Therefore in general and for real-time traffic, which are sensitive to latency, it is strongly recommended to coalesce at high load and not to coalesce at low load (i.e., use normal interruption). And hence there is a need to monitor traffic load and switch accordingly to either normal interruption or coalescing. The paper identified a crucial point of operation that can be used to determine overload condition (the saturation or cliff point). This point can be the switching point between normal interruption and coalescing.

Other important concluding remarks can also be made. First, when switching to interrupt coalescing at high load, a large value for the coalescing parameters should be chosen. Second, for traffic that can tolerate latency such as that of FTP, coalescing at both low and high loads becomes a practical scheme as it gives better throughput, CPU availability, and packet loss. Third, for non-real time traffic, utilizing time-based coalescing can be more practical than count-based coalescing as time-based coalescing imposes a limit on the coalescing delay at low load.

As a future work, we are currently in the process of modifying Linux kernel 2.6 to implement a hybrid scheme of normal interruption and coalescing taking into account all the recommendations made in this paper. Results of such implementation will be published in the near future.

References

[1] J. Mogul, and K. Ramakrishnan, "Eliminating Receive Livelock In An Interrupt-Driven Kernel," *ACM Trans. Computer Systems*, vol. 15, no. 3, August 1997, pp. 217-252.

[2] A. Indiresan, A. Mehra, and K. G. Shin, "Receive Livelock Elimination via Intelligent Interface Backoff," TCL Technical Report, University of Michigan, 1998.

[3] P. Druschel, "Operating System Support for High-Speed Communication," *Communications of the ACM*, vol. 39, no. 9, September 1996, pp. 41-51.

[4] P. Druschel, and G. Banga, "Lazy Receive Processing (LRP): A Network Subsystem Architecture for Server Systems," Proceedings Second USENIX Symposium on Operating Systems Design and Implementation, October 1996, pp. 261-276.

[5] R. Prasad, M. Jain, and C. Dovrolis, " Effects of Interrupt Coalescence on Network Measurements," Proceedings of Passive and Active Measurement (PAM) Workshop, France, April 2004.

[6] M. Zec, M. Mikuc, and M. Zagar, "Estimating the Impact of Interrupt Coalescing Delays on Steady State TCP Throughput", Proceedings of the 10th SoftCOM, October 2002.

[7] I. Kim, J. Moon, and H. Y. Yeom, "Timer-Based Interrupt Mitigation for High Performance Packet Processing, " Proceedings of 5th International Conference on High-Performance Computing in the Asia-Pacific Region, Gold Coast, Australia, September 2001.

[8] P. Shivan, P. Wyckoff, and D. Panda, “EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing,” Proceedings of SC2001, Denver, Colorado, USA, November 2001.

[9] C. Dovrolis, B. Thayer, and P. Ramanathan, “HIP: Hybrid Interrupt-Polling for the Network Interface,” *ACM Operating Systems Reviews*, vol. 35, October 2001, pp. 50-60.

[10] Alteon WebSystems Inc., “Jumbo Frames,” http://www.alteonwebsystems.com/products/white_papers/jumbo.htm

[11] A. Gallatin, J. Chase, and K. Yocum, "Trapeze/IP: TCP/IP at Near-Gigabit Speeds", Annual USENIX Technical Conference, Monterey, Canada, June 1999.

[12] C. Traw, and J. Smith, "Hardware/software Organization of a High Performance ATM Host Interface," *IEEE JSAC*, vol.11, no. 2, February 1993.

[13] C. Traw, and J. Smith, "Giving Applications Access to Gb/s Networking," IEEE Network, vol. 7, no. 4, July 1993, pp. 44-52.

[14] K. Salah and K. Badawi, "Evaluating System Performance in Gigabit Networks", The 28th IEEE Local Computer Networks (LCN), Bonn/Königswinter, Germany, October 20-24, 2003, pp. 498-505

[15] K. Salah, “Two Analytical Models for Evaluating Performance of Gigabit Ethernet Hosts with Finite Buffer”, International Journal of Electronics and Communications, Elsevier Science, In Press, *Available online 15 November 2005*.

[16] A. Law and W. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, 2nd Edition, 1991.

[17] J. White, “An Effective Truncation Heuristic for Bias Reduction in Simulation Output,” *Simulation Journal*, vol. 69, no. 6, December 1997, pp. 323-334

[18] K. Salah and K. El-Badawi, “Throughput and Delay Analysis of Interrupt-Driven Kernels under Poisson and Bursty Traffic”, International Journal of Computer Systems Science and Engineering, CRL Publishing, Accepted, 2005.